PROGRAMACION:

* Es una disciplina, es la ciencia de informática o la computación.
* Ciencia que se dedica al análisis y transformación de datos e información a partir del uso de computadoras.

PROGRAMA:

* Es una solución a un problema.
* Es un o un conjunto de ALGORITMOS.

ALGORITMOS:

* Es un o un conjunto de INSTRUCCIONES, que están seteadas o ordenadas de tal manera, que cuando se proceda a la ejecución, deberíamos llegar al RESULTADO DESEADO, o POS DE SOLUCIONAR UN PROBLEMA.

PROGRAMAR:

* Es enseñarle a la computadora.
* Va a ser el medio para enseñarle a la computadora los pasos que debe seguir para RESOLVER la PROBLEMA DEL USUARIO.
* PERO ANTES… de PROGRAMAR, debemos ser capaces de RESOLVER ESA PROBLEMA, para luego enseñarle a la computadora.

CAMINO BASICO DE LA PROGRAMACION

DATOS

PROCESAMIENTO

INFORMACION

DATOS:

* Es un o un conjunto de símbolos que no tiene un significado alguno.
* DAMOS datos.

PROCESAMIENTO:

* Se APLICAR un procesamiento de esos DATOS dados.

INFORMACION;

* Es un o un conjunto de símbolos que fue dado como dato, pero con un significado atribuido.
* Devolver información.

PENSAMIENTO DIVERGENTE:

* No hay una sola solución a un problema determinado.

CAPACIDAD DE ALMACENAMIENTO.

* BIT
* 1 BYTE (8 BIT)
* 1 KBYTE (1024 BYTE)
* 1 MBYTE (1024 KBYTE)
* 1 GBYTE (1024 MBYTE)
* 1 TERABYTE (1024 GBYTE)

RAM:

* Random Access Memory
* Todo aquello que se este ejecutando o procesando en el momento, esta en la memoria RAM guardada.
* Es VOLATIL, NO guarda información. MEMORIA TEMPORAL.
* SOPORTE DEL PROCESADOR DIARIO.

PROCESADOR:

* Es el cerebro.
* Conocido como Cpu o GPu.
* Es el que se encarga de mover toda la información y datos.

DISCO RIGIDO:

* MEMORIA A LARGO PLAZO.
* Donde se guarda toda la información.

DIAGRAMA DE FLUJO.

INICIAR

INGRESO DE DATOS

PROCESAMIENTO DE ESTOS DATOS INGRESADOS

EGRESO, SALIDA DE LA INFORMACION SOLICITADA

FINALIZACION

VARIABLE:

* Es un espacio en memoria dedicado o asignado por el sistema operativo.
* Donde se guardarán los datos que necesitemos.

LENGUAJE DE PROGRAMACION:

* Son la distinta forma de escribir un código.
* Permite la comunicación entre Personas y Computadoras.
* En el cual existen Lenguajes de Alto nivel o de Bajo nivel.

COMPILAR:

* Es la acción de transformar ese código que nosotros escribimos en un lenguaje que nosotros entendemos a un lenguaje denominado lenguaje de máquina.
* Lenguaje Binario (01)
* El lenguaje de mas bajo nivel, es el lenguaje Binario, el que entienden las maquinas.

SINTAXIS:

* Forma de escribir un código determinado dependiendo el lenguaje.

VARIABLE CONSTANTE:

* Son variables que no pueden variar.

Ejemplo:

|  |  |
| --- | --- |
| 1 | **const** **int** variable = **2**; |
|  |  |

PROGRAMACION ES SECUENCIAL:

* Siempre se ejecutará de arriba hacia abajo.

CUANDO DECLARO A UNA VARIABLE, ESTOY RESERVANDO ESPACIO EN MEMORIA.

INICIALIZAR, DAR UN VALOR INICIAL A DICHA VARIABLE.

LENGUAJE C# (SHARP)

* Lenguaje desarrollado por Microsoft.
* Visual Studio Code = editor de código.

Código INICIAL.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | **using** **System**;  **namespace** **tipos\_de\_datos**  {  **class** **Program**  {  **static** **void** **Main**(**string**[] args)  {      }  }  } |

Visual Studio Code.

* Usar dotnet run
* dotnet - -version
* Dotnet new console -o nombreDeArchivo - -framework net5.0
* Cd ..
* Cd nombreDeArchivo

Tipos de Datos Elementales:

* Int.
* Float.
* Char.
* Bool.

Espacio de memoria que ocupa cada Tipo de Dato.

* sbyte -128 to 127 (8bit)
* byte 0 to 255 (8bit)
* short -32768 to 32767 (16bit) (2byte)
* ushort 0 to 65535 (16bit) (2byte)
* int -2147483648 to -2147483647 ( 32bits) (4byte)
* float 6 a 9 digitos (4byte) (32bits)
* double 15 a 17 digitos (8byte) (64bits)
* decimal 28 a 29 digitos (16byte) (128bits)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **float** precio;  **decimal** precio1;  **double** precio2;  precio = **12.45f**;  precio1 = **12.45**m;  precio2 = **12** |

Asignación:

**int** n;

n = **2**;

// Asignación, primero

resolvemos lo de la derecha y

//luego asignamos un nuevo

valor a la variable.

n = n + n;

OPERADORES RELACIONALES:

* < , > , <= , >= , ¡= , ==

OPERADORES LOGICOS O CONCATENADORES LOGICOS.

* AND / && (doble ampersand).
* OR / || (doble pipe).
* NOT / ¡

CONDICIONALES:

* IF/ELSE/ ELSE IF.
* (valor / operador relacional / valor). << Es la condición, que puede ser True o False.
* En un bloque de código, solamente se puede sacar las llaves {}, si escribí solamente una línea de código.
* Siempre va a ver dos opciones, true o false, se bifurcan los caminos, y luego se vuelven a unir.
* Siempre tiene bloque de código, tanto en el IF, ELSE, ELSE IF, con apertura y cierre de llaves.

**namespace** **condicional\_if**

{

**class** **Program**

{

**static** **void** **Main**(**string**[] args)

{

**int** edad = **45**;

**int** edad2 = **20**;

**if** (edad > **21** ) {

**if** (edad > **30**){

Console.WriteLine("30% OFF");

}

**else**{

Console.WriteLine("21% OFF");

}

}

**else** {

Console.WriteLine("10% OFF");

}

Console.WriteLine("=================================");

**if** (edad2 > **30** )

Console.WriteLine("30% OFF");

**else** **if** (edad2 > **20**)

Console.WriteLine("21% OFF");

**else**

Console.WriteLine("10% OFF");

Console.WriteLine("Fin del programa");

CONTADORES:

* variable ++:
* Cuenta (1 ,2, …)

ACUMULADORES:

* Variable += variable
* Acumulada (2, +4, +8, +4…)

**int** promedio, edad = **40**, edad1 = **20**, edad2 = **30**;

// Podemos plantarlo de dos maneras , declarando a las variables la de arriba o la de abajo.

**int** contador = **0**;

**int** acumulador = **0**;

**if** (edad > **19**){

contador++;

acumulador += edad;

}

**if** (edad1 > **18**){

contador++;

acumulador += edad1;

}

**if** (edad2 > **20**){

contador++;

acumulador += edad2;

}

promedio = acumulador / contador;

Console.WriteLine("\nCantidad: " + contador +

"\nAcumulador: " + acumulador +

"\nPromedio: " + promedio+"\n");

Otro CONDICIONALES:

SWITCH (primo del IF)

* Evalúa el contento de una Variable.

**int** procesador, ram =**0**, disco;

**switch** (disco){

**case** **1**:

totalPagar += **300**;

componenteDisco = "1TB";

**break**;

**case** **2**:

Console.WriteLine("\nNo desea expandir el Disco Rigido.");

componenteDisco = "500GB";

**break**;

**default**:

Console.WriteLine("\nOpcion incorrecta.\nVuelva a intentarlo.");

**break**;

}

CICLO FOR.

* Permite que una instrucción se repita tantas veces que necesitemos, escribir una instrucción una sola vez, y que se ejecute varias veces.
* ITERAR, es dar una vuelta.
* Es un ciclo EXACTO, va a dar la cantidad de vueltas que le pidamos.

Tres factores para configurar el ciclo for:

* INICIALIZACION: (i = 0), dar un valor inicial a la variable que rige el ciclo.
* CONDICION: (i < 10), es la instrucción que determina si sigue iterando o no.
* INCREMENTO: (i++;), es la instrucción que se encarga de modificar el valor de la variable que rige el ciclo.

/\*

Hacer un programa para mostrar los números del 1 al 10. No se debe realizar ningún pedido de datos.

\*/

**for**( **int** i = **1**; i < **11**; i++){

Console.WriteLine(i);

}

CICLO WHILE:

* Es un ciclo INEXACTO, porque no le decimos la cantidad de iteraciones que debe dar, a diferencia del ciclo FOR.
* Itera igual que el ciclo FOR.
* Dependiendo la condición establecida se repetirá la instrucción tantas veces como fuese necesario.

**int** n;

**while** (n ¡= 0) {

Console.WriteLine(n);

}

DO WHILE:

* Es lo mismo que el While, solo que la instrucción al menos se ejecutara una vez, porque hace y luego pregunta (condición).

do {

} while (condición)

CICLOS COMBINADOS 0 ANIDADOS:

* Es usar dos o mas ciclos exactos o inexactos).
* Generalmente hay un LOTE Y UN SUBLOTE, un GRUPO Y UN SUBGRUPO.

/\*

Hacer un programa para ingresar 10 números.

El mismo debe analizar y mostrar por pantalla cuántos de esos números son primos.

\*/

Console.WriteLine("\nIngrese 10 numeros, y le mostrare cuantos son PRIMOS.\n");

**int** primos, conPrimos, conTotal = **0**;

**for** (**int** x = **0**; x < **10**; x++){

Console.Write("Ingrese el numero " + (x + **1**) + " :");

primos = **int**.Parse(Console.ReadLine());

conPrimos = **0**;

**for** (**int** y = **1**; y <= primos; y++){

**if**(primos % y == **0**){

conPrimos++;

}

}

**if** (conPrimos == **2**){

conTotal++;

}

}

Console.Write("\nCantidad de PRIMOS: " + conTotal + "\n");

CORTE DE CONTROL:

* Se agrupan en grupos, ejemplo Facturas A, B y C.
* Sirven para cargarse en grupo, pero deben estar ordenados, no puedo cagar grupo A, luego B, y volver al grupo A, si haría eso se rompería.
* Se usan Lote y Sublote.
* Se utilizan Dos Ciclos While, uno dentro del otro.
* Cambian cuando cambiamos de Registro, GRUPO A, a GRUPO B.

FUNCIONES:

* Es un pequeño programa, pequeño bloque de código o diagrama para resolver una sola cosa.
* Es un bloque de código que tiene un inicio y un final
* Todos los datos que recibe la función se denominan PARAMETROS o ARGUMENTOS.
* Cuando la función termina de trabajar con un PARAMETRO, RETORNA (RETURN) un Resultado.
* Las funciones van fuera de otras funciones.
* Las funciones NO piden número, sino que RECIBEN, y NO IMPRIMEN, SINO QUE RETORNAN (RETURN) UN RESULTADO.
* FUNCION = RECIBE >>> PROCESA>>> DEVUELVE.

Como están armadas las funciones:

CABECERA:

* TIPO DE FUNCION: int, float, char, string, etc.
* NOMBRE: que nombre va a tener la función.
* ARGUMENTO O PARAMETROS. (opcional).

CUERPO:

* DEFINICION / LOGICA.
* RETORNO DE UN VALOR / RESULTADO (opcional).

IMPORTANTE, LOS ARGUMENTO SE PUEDEN PASAR POR VALOR, O REFERENCIA.

* VALOR: es cuando envió una copia de mi argumento, y este no puedo ser modificado, pero si puede ser usado en la función.
* REFERENCIA: es cuando la función, puede manipular y modificar ese argumento, es como si le estaría pasando el original, y puede usarlo o hacer lo que desee. Envió la dirección de memoria de mi variable.

**namespace** **ejercicio1**

{

**class** **Program**

{

**static** **void** **Main**(**string**[] args)

{

/\*

Hacer una función llamada “producto” que reciba dos números enteros y que devuelva el producto de ambos.

Luego hacer un programa que pida el precio de un artículo y la cantidad vendida y muestre por pantalla el monto total a pagar.

Usar la función

\*/

**int** cantidadVendida, precio, montoPagar;

Console.Write("\nPRECIO PRODCUTO: " );

precio = **int**.Parse(Console.ReadLine());

Console.Write("");

Console.Write("CANTIDAD VENDIDA: ");

cantidadVendida = **int**.Parse(Console.ReadLine());

montoPagar = producto(precio, cantidadVendida);

Console.WriteLine("\nTOTAL A PAGAR: " + montoPagar);

}

**static** **int** **producto**(**int** a, **int** b)

{

**int** resultado = a \* b;

**return** resultado;

}

}

}

VECTORES (ARRAYS / ARREGLO)

* Una colección de elementos del mismo tipo (int, float,etc).
* Permite guardar muchos valores en la misma variable.
* Los vectores me permiten reservar espacio de memoria para mas de un elemento, ejemplo [10]
* Existe método burbuja, que permite ordenar el vector de mayor a menos o viceversa.

Int [ ] números = new int [10]

Vectores de caracteres:

* STRING ¡= CADENA DE CARACTERES ¡= VECTOR DE CHAR.
* Vector de CHAR: es un vector del tipo Char, y solo guardo caracteres, y termina en la última posición del índice.
* Cadena de CARACTERES: es un vector del tipo Char, en el cual también guardo caracteres, pero ordenándolo con un sentido de palabra o frase, y finaliza o pongo fin con \0, en el cual puede finalizar antes.

**using** **System**;

**namespace** **ejercicio1**

{

**class** **Program**

{

**static** **void** **Main**(**string**[] args)

{

/\*Hacer un programa que solicite 10 números enteros y los guarde en un vector.

Luego recorrer los elementos y determinar e informar cuál es el valor máximo y su posición dentro del vector.\*/

// 1= int max =0 , posicion = 0;

**int**[] vector = **new** **int**[**10**];

**int** max = vector[**0**], posicion = **1**;

Console.WriteLine("\nIngrese 10 numeros, y le dire cual es el VALOR MAXIMO y en que POSICION.\n");

Console.WriteLine("Numeros a Ingresar: \n");

**for** (**int** x = **0**; x < **10**; x++){

vector[x] = **int**.Parse(Console.ReadLine());

**if** (vector[x] > max){

max = vector[x];

posicion = x + **1**;

}

}

// 1= for (int y = 0; y < 10; y++){

// if (y == 0){

// max = vector[y];

// posicion = y + 1;

// }

// else {

// if (vector[y] > max){

// max = vector[y];

// posicion = y + 1;

// }

// }

// }

Console.WriteLine("\nValor Maximo: " + max + "\nPosicion: " + posicion);

FRAMEWORK:

* Conjunto Estandarizado.
* Estructura conceptual y tecnológica.
* Agrupa modelos y herramientas.
* Permite Optimizar desarrollos.
* En simple palabras, al escribir una gran cantidad de línea de código, puede aparecer errores que son difíciles de detectar o encontrar sin una organización adecuada,
* es donde acá aparecen los framework que nos :
  + ayuda a facilitar la escritura de código
  + , y a la organización de la misma,
  + ayudan a acelerar la tarea de programar, con la creación y organización de código, ganando tiempo escribiendo código o encontrando errores,
  + Autocompledo, o resaltado de sintaxis.
  + teniendo las herramientas listas para usar.

.NET FRAMEWORK:

* Es un conjunto de herramientas, tecnologías y servicios que nos facilita la construcción de varios tipos de APP.
* Plataforma de Desarrollo de Software: permite desarrollar y ejecutar nuestros desarrollos.
* Plataforma: son todos aquellos elementos que me sirven para construir algo. (Aplicaciones, escritorio, consola, etc)
* Lenguaje de programación: acepta C#, Visual Basic, C++, etc

.NET CORE:

* Es un framework igual que el .net framework, pero actualizado.
* Es gratis, se puede construir aplicaciones y comercializar sin trabas.
* Código abierto en el cual se puede compartir código y mejorarlo
* Es multiplataforma: Linux, Windown, Mac

VISUAL STUDIO COMMUNITY

* IDE: Integrated Development environment (entorno de Desarrollo integrado , ejemplo es una app ).
* Es un programa, que es donde vamos a escribir nuestro código y programar.
* Hermano mayor del Visual Studio Code.
* Y este es un entorno de desarrollo en el cual podemos trabajar con .Net

HAY QUE ACLARAR QUE NOSOTROS PROGRAMAMOS EN C# O HABLO EN ESE IDIOMA, CON .NET. QUE USO ESTA HERRAMIENTAS QUE ME AYUDA.

POO

* Es un paradigma: es una forma de hacer las cosas.

CLASE:

* Creamos un tipo de dato.
* Cuando creamos una clase voy a definir una estructura que va a tener algo y con esto tratamos de representar la realidad.
* Es un elemento de la realidad que va a manipular la APP.
* Class nombre >> tipo de dato (lo que represento de la realidad a lo digital)
* {bloque de código}, van a estar los ATRIBUTOS o PROPIEDADES, son variables que nos permite guardar información de esa clase, ejemplo persona, ATRIBUTO DE CLASE.
* METODOS, sería la acción aplicada al objeto.
* Decimos que los métodos son funciones, pero desde las clases se denominan métodos.
* ATRIBUTOS O PROPIEDADES, son característica de una clase, cuando construimos una clase definimos una serie de atributos, o sus propiedades y una serie de métodos,

PERO CUANDO INSTACIAMOS AL OBJETO, los atributos acá serian la información del objeto, esta información hace al estado de un objeto, el estado está dado por la información que tiene el objeto.

QUE LOS METODOS, AQUÍ SE DENOMINAN COMPORTAMIENTOS, que son los métodos definidos en la clase.

CONSTRUCTORES:

* Permite inicializar los objetos dándole la información a cada atributo, por única vez.

OBJETO:

* Representación de la realidad, Y se crea el objeto, recién cuando se instancia la clase dicha, no se le dice método, aquí son comportamientos.

Encapsulamiento:

* Variables que no pueden ser accedidas desde el exterior de forma directa, pero podemos acceder de forma “indirecta” con el property {get, set}
  + Get y Set son métodos de una clase, que uno me permite ver el valor, y el otro modificar.
* Es una barrera entre el usuario y los atributos.

**namespace** **sobreCargaMetodo**

{

**internal** **class** **Persona**

{

**private** **string** nombre;

**private** **int** edad;

**public** **Persona**(**string** nombre, **int** edad)

{

**this**.nombre = nombre;

**this**.edad = edad;

}

**public** **string** Nombre

{

**get** { **return** nombre; }

}

**public** **int** Edad

{

**get** { **return** edad; }

}

// Metodo

**public** **string** **saludar**()

{

**return** "Hola como estas " + nombre;

}

// Este es sobrecarga de metodo , con el metodo saludar.

**public** **string** **saludar**(**string** nombre)

{

**return** "Hola " + nombre + " como estas?, mi nombre es " + **this**.nombre;

}

}

}

namespace sobreCargaMetodo

{

internal class Program

{

static void Main(string[] args)

{

Persona usuario1 = new Persona("Fernando", 34);

Console.WriteLine(usuario1.saludar());

Console.WriteLine(usuario1.saludar("Cacho"));

Console.ReadKey();

}

}

}

MODIFICAR UN SET.

namespace ejemplo1

{

internal class Articulo

{

public int CodigoArticulo { get; set; }

public float Precio { get; set; }

private int codMarca;

public int CodigoMarca

{

get { return codMarca; }

set

{

if (value > 0 && value < 11)

codMarca = value;

else

codMarca = -1;

}

}

}

namespace constructorYdestreuctor

{

internal class Botella

{

//Atributos del obejto

private string color;

private string material;

private int capacidad;

private int cantidadActual;

private float precioPorRecarga;

//Constructor, declarar esos atributos por un unica vez, cuando instanciamos el obejto..

public Botella(string color, string material)

{

this.color = color;

this.material = material;

capacidad = 100;

cantidadActual = 0;

precioPorRecarga = 50;

}

//Property (para conseguir el dato de la capacidad con el get)

public int Capacidad

{

get { return capacidad; }

}

public int CantidadAtual

{

get { return cantidadActual; }

set {cantidadActual = value; }

}

public float PrecioPorRecarga

{

get { return precioPorRecarga; }

set {precioPorRecarga = value; }

}

//Metodo: la funcionalidad o el comportamiento.

public float recargar()

{

if (cantidadActual > 0)

{

int diferencia = capacidad - cantidadActual;

float montoApagar = diferencia \* precioPorRecarga / capacidad;

cantidadActual += diferencia;

precioPorRecarga = montoApagar;

return precioPorRecarga;

}

cantidadActual = 100;

return precioPorRecarga;

}

}

}

SOBREESCRIBIR METODO.

namespace SobreEscribirMetodos

{

class Animal

{

private string nombre;

public string Nombre { get; set; }

//ACA ESTOY SOBRE ESCRIBIENDO EL METODO TO\_STRING DE LA CLASE OBJECT , DE LA CLASE PADRE DE TODOS.

// con el Override SobreEScribo un Metodo. ejemplo ToString.

public override string ToString()

{

return "Animal " + Nombre;

}

// para Override con otras jerarquias mas bajas .

public virtual string comunicarse()

{

return "Ruido...ruido...";

}

}

}

namespace SobreEscribirMetodos

{

class AnimalDomestico : Animal

{

}

}

namespace SobreEscribirMetodos

{

internal class Perro : AnimalDomestico

{

public override string comunicarse()

{

return "Guau Guau";

}

}

}

namespace SobreEscribirMetodos

{

class AnimalSalvaje : Animal

{

namespace SobreEscribirMetodos

{

internal class Program

{

static void Main(string[] args)

{

AnimalDomestico animalDomes1 = new AnimalDomestico();

Perro perro1 = new Perro();

perro1.Nombre = "Pepito";

animalDomes1.Nombre = "Juan Carlos";

// Hacer una LISTA de animales MEZCLADOS, en esta lista puedo agregar cualquier tipo de ANIMAL .

List <Animal> animales = new List<Animal>();

animales.Add(animalDomes1);

animales.Add(perro1);

animales.Add(new Tigre());

foreach (Animal animal in animales)

{

Console.WriteLine(animal.comunicarse());

}

// hago la sobre escritura del metodo ToString(); de la Clase Objects , la clase padre y madre de todas.

Console.WriteLine(animalDomes1.ToString());

Console.WriteLine(animalDomes1.comunicarse());

Console.WriteLine(perro1.comunicarse());

// Puedo pasar una variable pero1 a animal con una variable x

Animal a1 = perro1; //pero a1 va tener todas las propiedades o atributos y metodos de que tiene Animal,

//si hay algo que se creo en PERRO un atributo nuevo, en Animal no se podra ver. Por que no conoce

//ese atributo, siempre va de arribar hacia abajo, jerarquicamente.

Perro perro8 = (Perro)a1; //poniendo (Perro) aclaro y le confirmo al compilador que ahi hay un perro.

Console.WriteLine(perro8.Nombre);

Console.ReadKey();

}

}

}

HERENCIA.

* Herencia “reutilizamos códigos generalmente”
* Es una relación entre clases (Familia o jerarquía de clases).
* La clase padre de todas las class es la clas OBJECT, y todas tienen alguna característica de esta clase de forma implícita.
* Adquiere sus características (atributo, propiedad y métodos).

namespace Herencia

{

class Persona

{

private string nombre;

private string apellido;

private int legajo;

}

}

namespace Herencia

{

class Lider : Persona

{

}

}

namespace Herencia

{

class Developer : Persona

{

}

}

VALORES Y REFERENCIAL.

* Valores: variables comunes que tienen valores comunes, de algún topo ( char , int, float, etc).
* Y la Referencial son los objetos.
* Esto se separa el almacenamiento en memoria los valores que declara una variable en una sección de memoria, y los objetos referenciales, en todas secciones de memoria diferentes.

Persona p1(se genera una variable) (declaro una variable) = new Persona (); (genera objeto) (instancio objeto), entonces v1 hace referencia al objeto.

Garbage collector, elimina la basura o lo que no se usa.,

COLECCIONES.

* Es dinámica, no tengo que definir el tamaño de esa colección elementos de un tipo.
* Es un objeto.
* Tipos de colecciones: SET, MAPA O DICCIONARIO, LIST.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Colecciones

{

internal class Program

{

static void Main(string[] args)

{

Vechiculo v1 = new Vechiculo();

Camioneta c1 = new Camioneta();

Camioneta c2 = new Camioneta();

Camioneta c3 = new Camioneta();

c1.Color = "Amarillo";

c2.Color = "Verde";

c3.Color = "Blanca";

List<Camioneta> listaCamionetas = new List<Camioneta>();

listaCamionetas.Add(c1); // acabo de agregar un elemento (objecto c1) a la listaCamionetas.

listaCamionetas.Add(c2);

listaCamionetas.Add(c3);

// FOR EACH permite recorrer una coleccion.

foreach (Camioneta camioneta in listaCamionetas)

{

Console.WriteLine("COLOR: " + camioneta.Color);

}

Console.WriteLine("La cantidad de CAMIONETAS ES: " + listaCamionetas.Count); // Contamos la cantidad de objetos o elementos.

listaCamionetas[0].Color = "Negra";// Cambio de color, a traves de la Lista.

Console.WriteLine("El color es: " + listaCamionetas[0].Color); // Buscar el elementos , y pedirle el Color.

listaCamionetas.Remove(c3);

Console.WriteLine("La cantidad de CAMIONETAS ES: " + listaCamionetas.Count);

Console.ReadKey();

}

}

}

LISTA:

* Fuertemente Tipado (debemos poner de que tipo de dato es).
* Es genérico porque puedo usar cualquier tipo de dato.
* Se acomodan en el orden que uno los va cargando (add)

SOBREESCRITURA DE METODO.

* Un método que voy a redefinir, y está en otra clase, en un método que hereda de mi clase base. Ejemplo toString();

namespace SobreEscribirMetodos

{

class Animal

{

private string nombre;

public string Nombre { get; set; }

//ACA ESTOY SOBRE ESCRIBIENDO EL METODO TO\_STRING DE LA CLASE OBJECT , DE LA CLASE PADRE DE TODOS.

// con el Override SobreEScribo un Metodo. ejemplo ToString.

public override string ToString()

{

return "Animal " + Nombre;

}

// para Override con otras jerarquias mas bajas .

public virtual string comunicarse()

{

return "Ruido...ruido...";

}

}

namespace SobreEscribirMetodos

{

class AnimalDomestico : Animal

{

}

}

namespace SobreEscribirMetodos

{

internal class Perro : AnimalDomestico

{

public override string comunicarse()

{

return "Guau Guau";

}

}

}

EN EL EJEMPLO ANTERIOR SE DA, EL POLIMORFISMO:

* La característica que tienen los objetos frente a un mismo estimulo comportarse diferente manera.

INTERFAZ.

* Es un contrato que deben cumplir para que no haya anomalía, o mala relación entre ambas partes.
* Una INTERFAZ es una especie de contrato que se va a implementar en alguna CLASE, y la CLASE debe cumplir con ese CONTRATO.
* Puede definir comportamiento.
* Una INTERFAZ solamente va a tener las FIRMAS (tipo, nombre y parámetros)
* Siempre se dice que se IMPLEMENTA, y NO se HEREDA.
* Una CLASE puede IMPLEMENTAR varias interfaces.
* Permite solamente OBJETOS.

namespace Interfaz

{

internal class Animal

{

}

}

namespace Interfaz

{

internal class AnimalDomestico : Animal

{

}

}

namespace Interfaz

{

internal class Canario : AnimalDomestico, FlyAble

{

public string volar()

{

return "Vuela como un canario...";

}

}

}

namespace Interfaz

{

interface FlyAble

{

// todas que tengan la interfaz flyable, deberan implementar el metodo volar (SI O SI)

string volar();

}

}

namespace Interfaz

{

internal class Program

{

static void Main(string[] args)

{

// para crear una lista con animales que solamente pueden volar

// cuando declaro el tipo, pongo la interfaz.

List<FlyAble> listaVoladores = new List<FlyAble>();

listaVoladores.Add(new Canario());

// en la arquitectura de clase que he disenado no puedo agregar otro tipo de animal que no sepa volar..

}

}

}

ASOCIACION.

* Es un tipo de relación entre clases.
* Existe COMPOSICION, Y LA AGREGACION.
* Las ASOCIACIONES se lee como TIENE y la herencia como ES

namespace Asociacion

{

class Auto : Vehiculo

{

private string modelo;

private int anio;

private string color;

public Auto()

{

Chasis = new Chasis(); // debe nacer con este atributo. // composicion

}

public string Modelo { get; set; }

public string Color { get; set; }

public int Anio { get; set; }

// aca tendremos un objeto del tipo motor, COMO PROPIEDAD O ATRIBUTO DEL AUTO.

// Composicion. Si el objeto deben nacer si o si con EL ATRIBUTO DE ESE OBJETO seria composicion

public Chasis Chasis { get; }// Debe nacer con un chasis, en el cual no podria modificarles sus atributos.

// Agregacion. Si el objeto puede nacer IGUAL sin EL ATRIBUTO, DE ESE OBJETO agregacion

public Motor Motor { get; set; }

}

}

namespace Asociacion

{

internal class Chasis

{

}

}

namespace Asociacion

{

class Motor

{

private string marca;

public string Marca{get; set;}

}

}

namespace Asociacion

{

internal class Program

{

static void Main(string[] args)

{

Auto a1 = new Auto();

a1.Motor = new Motor(); // Esto es Agregacion porque le puede agregar el motor despues.

}

}

}

TIPOS DE CLASES ABSTRACTA.

namespace TipoDeClasesAbstractaYmas

{

abstract class Persona

{

// Si NO! necesito crear una instancia persona(), y no quiero que lo permita mi Modelo.

//y lo que puedo hacer es una Clase Abstracta, y de esta manera no puedo crear una INSTANCIA PERSONA,

//pero si las que heredan de Persona. < CONCLUSION se crea la Clase Abstracta para no poder ser instanciada >

public int Legajo { get; set; }

public string Nombre { get; set; }

public string Apellido { get; set; }

//Podemos tener otro modificador que se llama SEALED < Directamente NO me permite HEREDAR, y solamente podre instanciar

//esa Class >.

// otro es el STATIC

}

}

namespace TipoDeClasesAbstractaYmas

{

// otro es el STATIC, la clase no puede heredar, si la clase es STATIC , los metodos deben ser STATIC

// No se puede instanciar una clase STATIC, PUEDO USARLA SIN CREAR UNA INSTANCIA.

// No se debe crear tantas clases static, porque cuando abrimos el programa, ya se estan ejecutando, por el mero hecho

// que no debemos instanciarlas.

static class Lider

{

public static string saludo()

{

return "Hola...";

}

}

}

namespace TipoDeClasesAbstractaYmas

{

class Program

{

static void Main(string[] args)

{

// Console es una Class que tiene un metodo static., porque nadie la invoca a esa clase.

Console.WriteLine(Lider.saludo());

Console.ReadLine();

}

}

}

MODELO DE DOMINIO.

* Se usa UML (Lenguaje unificado de **modelado**), se trata de un gráfico de clases o esquemas
* Ciclo del sistema: Cual es la primera etapa, la segunda, etc. Generalmente se empieza por bases de datos, pero puedo variar.

BASE DE DATOS.

* PERSISTIR: es la acción de guardar un o una series de objetos o datos en una BASE DE DATOS.
* BASE DE DATOS: lugar donde vamos a guardar información, también denominado sistemas de baso de datos, y además de guardar datos nos ayuda con el mantenimiento.

TIPOS DE BASE DATOS.

* RELACIONALES: (basada en SQL), usa SQL , que es un lenguaje de consulta, se basa en tablas y relaciones.
* NO RELACIONALES: (basada en NoSQL), no usan SQL, y no están basadas en tablas o relaciones, y están orientadas a DOCUMENTOS O ARCHIVOS.

Utilizamos SQL Server Management , que esto nos va a permitir a acceder a los servicios que nos provee el motor, serie como un IDE, pero para sistemas de DB.

SQL es LENGUAJE ESTRUCTURA DE CONSULTA.

Consultas básicas que haremos, LEER, INSERTAR, MODIFICAR Y ELEMINAR DATOS.

PARA CONECTAR A LAS BASE DE DATOS DESDE .NET USAMOS UN LIBRERÍA DENOMINADA ADO.NET, HAY dos modelos CONECTADO o DESCONECTADO

REFACTORIZAR = es reordenar o reestructurar las soluciones, ordenándolas en capas

BASES DE DATOS (DB, SISTEMA DE BASES DE DATOS)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Data.SqlClient;

namespace pokemonWinForm

{

class PokemonNegocio

{

public List<Pokemon> Listar()

{

List<Pokemon> listaPokemones = new List<Pokemon>();

// Permite conectarme

SqlConnection conexion = new SqlConnection();

// Para realizar acciones con esa conexion

SqlCommand comando = new SqlCommand();

// y para guardar esa informacion , y alojarlo es...,

// y no se crea un objeto, porque cuando leo ese dato, voy a tener una instacia de un obejto...

SqlDataReader lector;

try

{

// este es conector de base de dato local, si fuera remote se colocaria un IP.

//("server = DESKTOP-KSD96OA \\SQLEXPRESS";) DESKTOP-KSD96OA = esto se puede reemplazar con un . (punto) , hace alusion a tu pc.

// luego debemos poner a que DB nos vamos a conectar (database)

// y por ultimo como me voy a conectar (para conectarme uso window authentication o sql authentication)

conexion.ConnectionString = "server = .\\SQLEXPRESS; database = POKEDEX\_DB; integrated security = true";

// aca debemos realizar la accion.

comando.CommandType = System.Data.CommandType.Text;

comando.CommandText = "Select Numero, Nombre, Descripcion From POKEMONS";

// para luego ejectuar este comando con esa conexion.

comando.Connection = conexion;

//Luego necesito abrir esa conexion.

conexion.Open();

// hago una lectura de eso..., y tengo los datos en la variable "lector".

lector = comando.ExecuteReader();

while (lector.Read())

{

// me creo un aux de tipo pokemon, para ir guardndo la informacion del lector.

// ya que si tiene informacion, el puntero se va a posicion en la primera fila,

// y va a dar los datos de esa primera fila.

Pokemon aux = new Pokemon();// en cada vuelta se crea una nueva instancia de ese objeto.

// para ver que get conseguir, debo posicionarme sobre la variable, en este caso numero

//y ver que type date es, y de ahi buscar getint, 16 es short, 32 int, 64 long.

aux.Numero = lector.GetInt32(0);// luego deberia poner el indice (0), que si vemos

//comando.CommandText = "Select Numero indice=0, Nombre indice=1, Descripcion indice=2From POKEMONS";

aux.Nombre = (string)lector["Nombre"];//<< tambien lo podemos hacer de esta manera

aux.Descripcion = (string)lector["Descripcion"];

listaPokemones.Add(aux);

}

return listaPokemones;

}

catch (Exception ex)

{

throw ex;

}

finally

{

conexion.Close();

}

}

}

}

WEB FORM.

Entorno WEB (Arquitectura WEB)

CLIENTE >> PETICION>> SERVIDOR >> RESPUESTA >> CLIENTE

Cliente hace una Petición al Servidor por medio de una URL, realiza una respuesta.

El CLIENTE seria el NAVEGADOR y el SERVIDOR, se le denomina la NUBE ( aquí es donde alojamos nuestra APP).

Ya no es de Forma local.

HTTP >> es un protocolo de comunicación, son pautas a seguir para realizar una comunicación eficaz (hablar el idioma dependiendo el país)

Lenguaje de la WEB son HTML, JS, CSS.

HTML

* HYPER TEXT MARKUP LANGUAJE.
* No requiere compliacion, es un lenguaje interpretado.
* Usamos editor de código, pero no es necesario, podríamos usar un bloc de notas.
* Es basado en etiquetas o tags, en el cual tienen apertura y cierre.
* Es documento estructurado.

Html, head, body etc.

CSS

* Le damos estilo o formato al HTML, digamos a esos tags.
* Seria un lenguaje de configuración.
* <style>
* Se puede colocar en línea (in line).
* Efecto cascada.

BASE DE DATOS

Por Stored Procedure (procedimiento almacenado) para Agregar

Insert into

Seria ejemplo

create procedure storedProcedureAltaPokemon

@numero int,

@nombre varchar(50),

@descripcion varchar(50),

@urlImagen varchar (300),

@idTipo int,

@idDebilidad int

as

insert into POKEMONS values (@numero, @nombre, @descripcion, @urlImagen, @idTipo, @idDebilidad, null, 1)

<<<lo hicimos de esta manera , porque completamos todos los datos de las columnas>> sino seris nombre columna (values) y el valor que le damos a esa columna

storedProcedureAltaPokemon 1,'','','',1,1,null, (para completarlo, directamente le damos el valor)

Modificar en DB con STORED PROCEDURE

create procedure storedModificar

@Numero int,

@Nombre varchar(50),

@Descripcion varchar(50),

@UrlImagen varchar(300),

@IdTipo int,

@IdDebilidad int,

@Id int

as

update POKEMONS set Numero= @Numero, Nombre= @Nombre, Descripcion = @Descripcion, UrlImagen= @UrlImagen, IdTipo = @IdTipo , IdDebilidad = @IdDebilidad where Id = @Id